Jack Girard
ECE 637
Lab 7: Image Restoration

1. Minimum Mean Square Error (MMSE) Linear Filters

1. The four original images *img14g.tif*, *img14bl.tif*, *img14gn.tif*, and *img14sp.tif*

Figure 1: img14g.tif

Figure 2: img14bl.tif

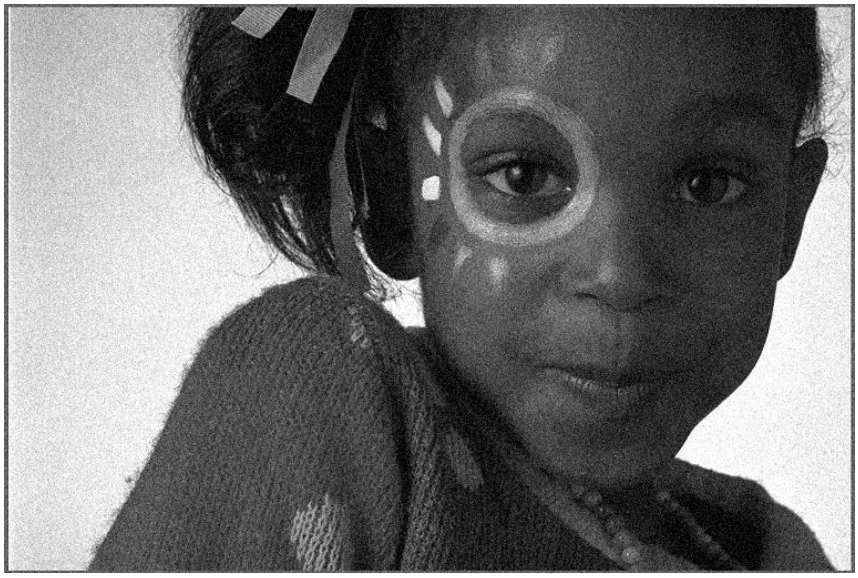

Figure 3: img14gn.tif

Figure 4: img14sp.tif

2. The output of the optimal filtering for the blurred image and the two noisy images

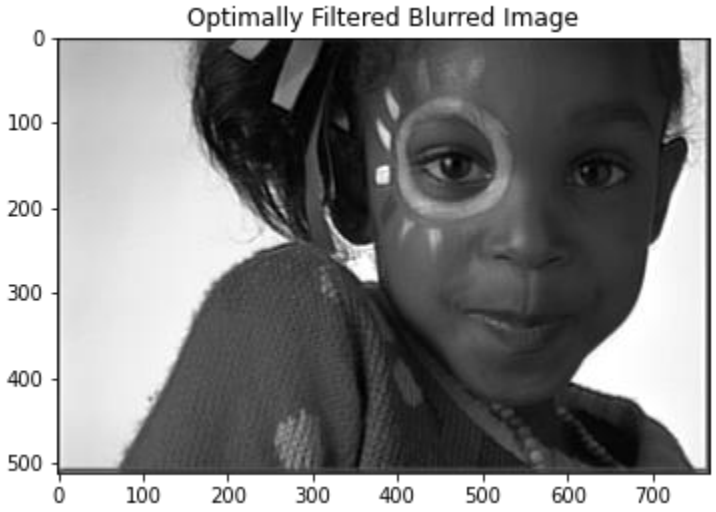

Figure 5: Blurred restoration

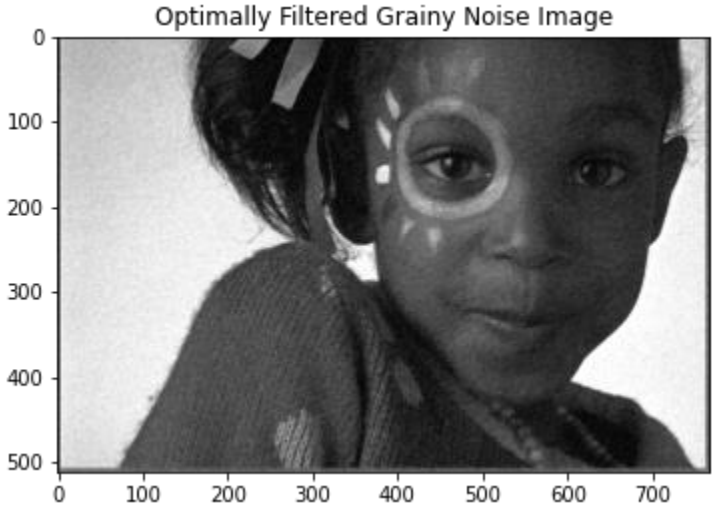

Figure 6: Grainy Restoration

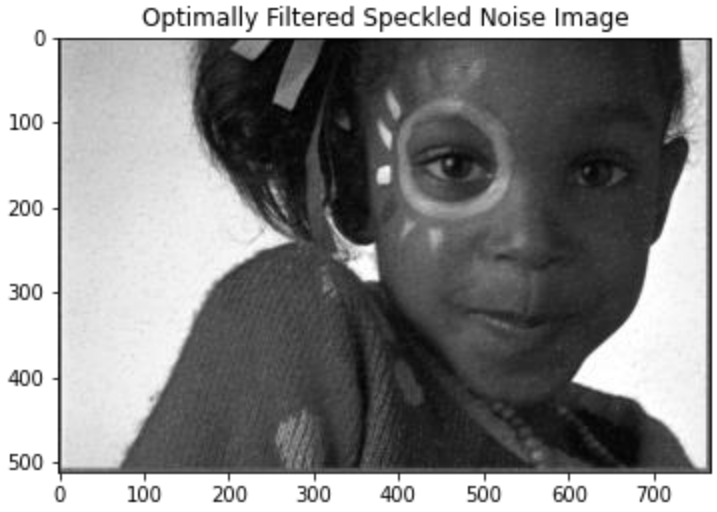

Figure 7: Speckled Restoration

3. The MMSE filters that were computed for the blurred image and the two noisy images

Blurred Image (*img14bl.tif*) Filter (Sum of θ* = 1.0045):

$\theta^* =$

```
[[-0.15004272 -2.20316248 -1.53751595  0.132943    0.55777773  1.17828702 -0.39453129]
 [ 2.13349788  2.79318104  0.92583732 -0.32647502  1.57534302  0.06452982  0.01833554]
 [-0.37621506 -1.7785595  -0.50835566  1.74945865 -1.47907556 -1.04886715  0.05890131]
 [-0.2209119   0.65281846 -0.01923163  0.04764148 -1.50344027 -0.93024642 -2.39822353]
 [-2.51493618 -0.78082127  0.83805166 -2.40253735  0.08848512  2.54612612 -0.61383685]
 [-0.5024801   4.65451482  0.35021105 -2.45114035  1.78221221  5.45638826  1.11487593]
 [ 0.21506558  0.63355317 -0.43247633 -3.73569145 -1.21106988  4.53496933 -3.57861374]]
```

Grainy Noisy Image (*img14gn.tif*) Filter (Sum of θ* = 1.0082):

$\theta^* =$

```
[[-0.02985835 -0.00114814  0.01621721 -0.01882117  0.00834596  0.01681519 -0.02012423]
 [-0.02547792  0.01879946  0.00257666 -0.0467791  -0.0076382  -0.02682051  0.0079573 ]
 [ 0.0175498   0.01970885  0.00225612  0.04266198  0.03079341 -0.00767855  0.01380388]
 [-0.01904948  0.00742443  0.0063532  -0.00808864 -0.02631122  0.01837885  0.04115626]
 [-0.01327874  0.01474441  0.01374941 -0.01074971 -0.0100289  -0.00147149  0.0588481 ]
 [-0.03977672 -0.02348193 -0.04029758 -0.03639427 -0.03256087  0.03986435  0.17840736]
 [ 0.03629389  0.03850956  0.05919588  0.05903991  0.08974773  0.18335814  0.41151886]]
```

Speckled Noisy Image (*img14sp.tif*) Filter (Sum of θ* = 1.0054):

$\theta^* =$

```
[[-0.03008728 -0.0322477   0.0213336  -0.04465128 -0.01242194  0.01097982  0.00387093]
 [-0.01064806  0.00622445  0.02520621  0.0096052  -0.02261044 -0.04280791 -0.00317061]
 [-0.00438541  0.0190518   0.0090709   0.03551449  0.01880543  0.03731365  0.03801995]
 [ 0.00232254 -0.03511802  0.00973322 -0.00241192 -0.00890723 -0.04438832 -0.00998791]
 [-0.00913994  0.00173587  0.00046863 -0.01861896 -0.01917184  0.05496598  0.08864654]
 [-0.02939398 -0.03493552 -0.02008879 -0.03894411 -0.00201718  0.04792006  0.17068996]
 [ 0.0559715   0.04983432  0.07536619  0.01987913  0.08651964  0.13957389  0.44292771]]
```

2. Weighted Median Filtering

1. Results of median filtering

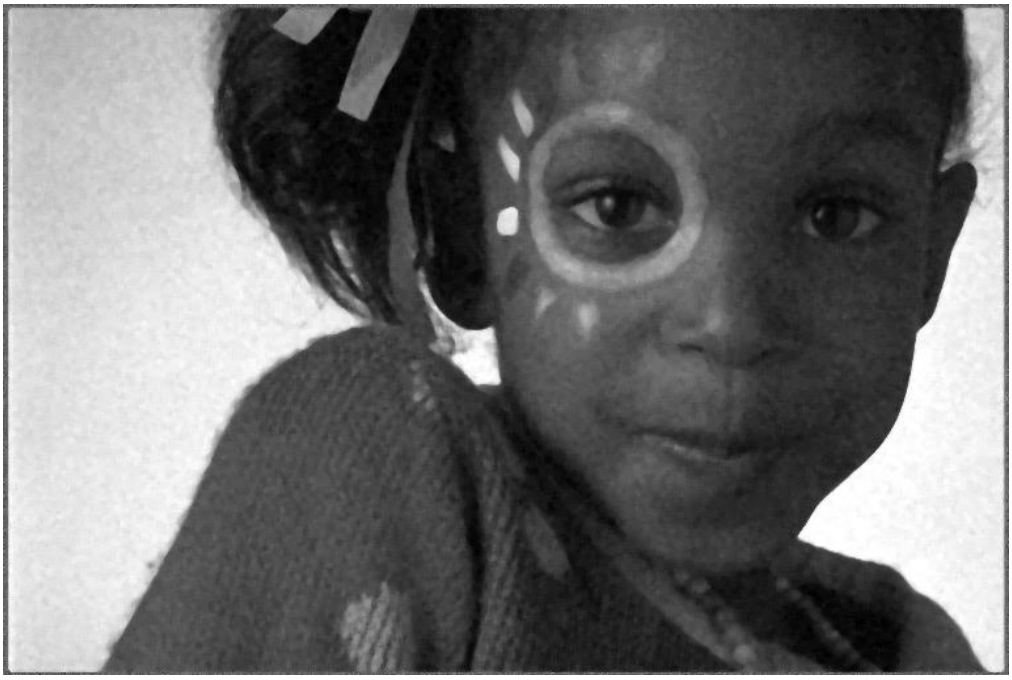

Figure 8: Weighted Median Filtered Result for img14gn.tif

Figure 9: Weighted Median Filtered Result for img14sp.tif

2. Attached C Code → *See Next Page*

```c

#include <math.h>
#include <string.h>

#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
#include "defs.h"       // header file that contains the necessary function prototypes

void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, output_img;
  unsigned int** filtered;
  int32_t i,j;

  if ( argc != 2 ) error( argv[0] );

  /* open image file */
  if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file %s\n", argv[1] );
    exit ( 1 );
  }

  /* read image */
  if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", argv[1] );
    exit ( 1 );
  }

  /* close image file */
  fclose ( fp );

  /* check the type of image data */
  if ( input_img.TIFF_type != 'g' ) {
    fprintf ( stderr, "error:  image must be grayscale\n" );
    exit ( 1 );
  }

  /* set up structure for output achromatic image */
  /* to allocate a full color image use type 'c' */
  get_TIFF ( &output_img, input_img.height, input_img.width, 'g' );

  /* call filterImage function which loops through each */
  /* pixel and applies weighted median filtering */
```

```
49    filtered = filterImage(input_img);
50
51    /* assign each element of filtered array to the */
52    /* monochrome data of the output image TIFF object */
53    for (i = 0; i < input_img.height; i++) {
54        for (j = 0; j < input_img.width; j++) {
55            output_img.mono[i][j] = filtered[i][j];
56        }
57    }
58
59    /* free data from filtered variable */
60    free_img((void*)filtered);
61
62    /* open green image file */
63    if ( ( fp = fopen ( "filtered_output.tif", "wb" ) ) == NULL ) {
64      fprintf ( stderr, "cannot open file green.tif\n");
65      exit ( 1 );
66    }
67
68    /* write green image */
69    if ( write_TIFF ( fp, &output_img ) ) {
70      fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
71      exit ( 1 );
72    }
73
74    /* close green image file */
75    fclose ( fp );
76
77    /* de-allocate space which was used for the images */
78    free_TIFF ( &(input_img) );
79    free_TIFF ( &(output_img) );
80
81    return(0);
82  }
83
84  void error(char *name)
85  {
86      printf("usage:  %s  image.tiff \n\n",name);
87      printf("this program reads in a 24-bit color TIFF image.\n");
88      printf("It then horizontally filters the green component, adds noise,
          \n");
89      printf("and writes out the result as an 8-bit image\n");
90      printf("with the name 'green.tiff'.\n");
91      printf("It also generates an 8-bit color image,\n");
92      printf("that swaps red and green components from the input image");
93      exit(1);
94  }
95
96  unsigned int calculateWMAtLocation(struct TIFF_img input, int xpos, int
```

```
      ypos) {
 97     /* initialize necessary variables & arrays */
 98     int temp, i, j, k, istar, weightedMedian, sum1, sum2;
 99     unsigned int input_pixels[25];
100     unsigned int weightingFactors[25] = { 1, 1, 1, 1, 1,
101     1, 2, 2, 2, 1,
102     1, 2, 2, 2, 1,
103     1, 2, 2, 2, 1,
104     1, 1, 1, 1, 1 };
105
106     /* populating input_pixels array from input TIFF object */
107     k = 0;
108     for (i = xpos - 2; i < xpos + 3; i++) {
109         for (j = ypos - 2; j < ypos + 3; j++) {
110             input_pixels[k] = input.mono[i][j];
111             k++;
112         }
113     }
114
115     /* apply brief sorting algorithm for both input pixels */
116     /* and corresponding weighting factors */
117     for (i = 0; i < 25; i++) {
118         for (j = i + 1; j < 25; j++) {
119             if (input_pixels[i] < input_pixels[j]) {
120                 temp = input_pixels[i];
121                 input_pixels[i] = input_pixels[j];
122                 input_pixels[j] = temp;
123
124                 temp = weightingFactors[i];
125                 weightingFactors[i] = weightingFactors[j];
126                 weightingFactors[j] = temp;
127             }
128         }
129     }
130
131     /* initialize summing variables */
132     sum1 = 0;
133     sum2 = 0;
134     for (i = 0; i < 25; i++) {
135         sum2 += sum2 + weightingFactors[i];
136     }
137
138     /* determining value for weighted median by incrementing */
139     /* i* and using Equation 7 from the assignment handout */
140     istar = 0;
141     while (sum1 < sum2) {
142         sum1 = 0;
143         sum2 = 0;
144         for (i = 0; i <= istar; i++) {
```

```
145             sum1 += sum1 + weightingFactors[i];
146         }
147         for (i = istar + 1; i < 25; i++) {
148             sum2 += sum2 + weightingFactors[i];
149         }
150 
151         if (sum1 >= sum2) {
152             weightedMedian = input_pixels[istar];
153         }
154         else {
155             istar++;
156         }
157     }
158     return weightedMedian;
159 }
160 
161 unsigned int** filterImage(struct TIFF_img input) {
162     /* initializing looping variables and 2D storage array */
163     int i, j;
164     unsigned int** filtered = (unsigned int**)get_img(input.width,
        input.height, sizeof(unsigned int));
165 
166     /* loop through each pixel of input image and apply weighted */
167     /* median filtering if proper window can be constructed */
168     for (i = 0; i < input.height; i++) {
169         for (j = 0; j < input.width; j++) {
170             if (i >= 2 && i < input.height - 2 && j >= 2 && j <
                input.width - 2) {
171                 filtered[i][j] = calculateWMAtLocation(input, i, j);
172             }
173             else {
174                 filtered[i][j] = input.mono[i][j];
175             }
176         }
177     }
178     return filtered;
179 }
180 
```